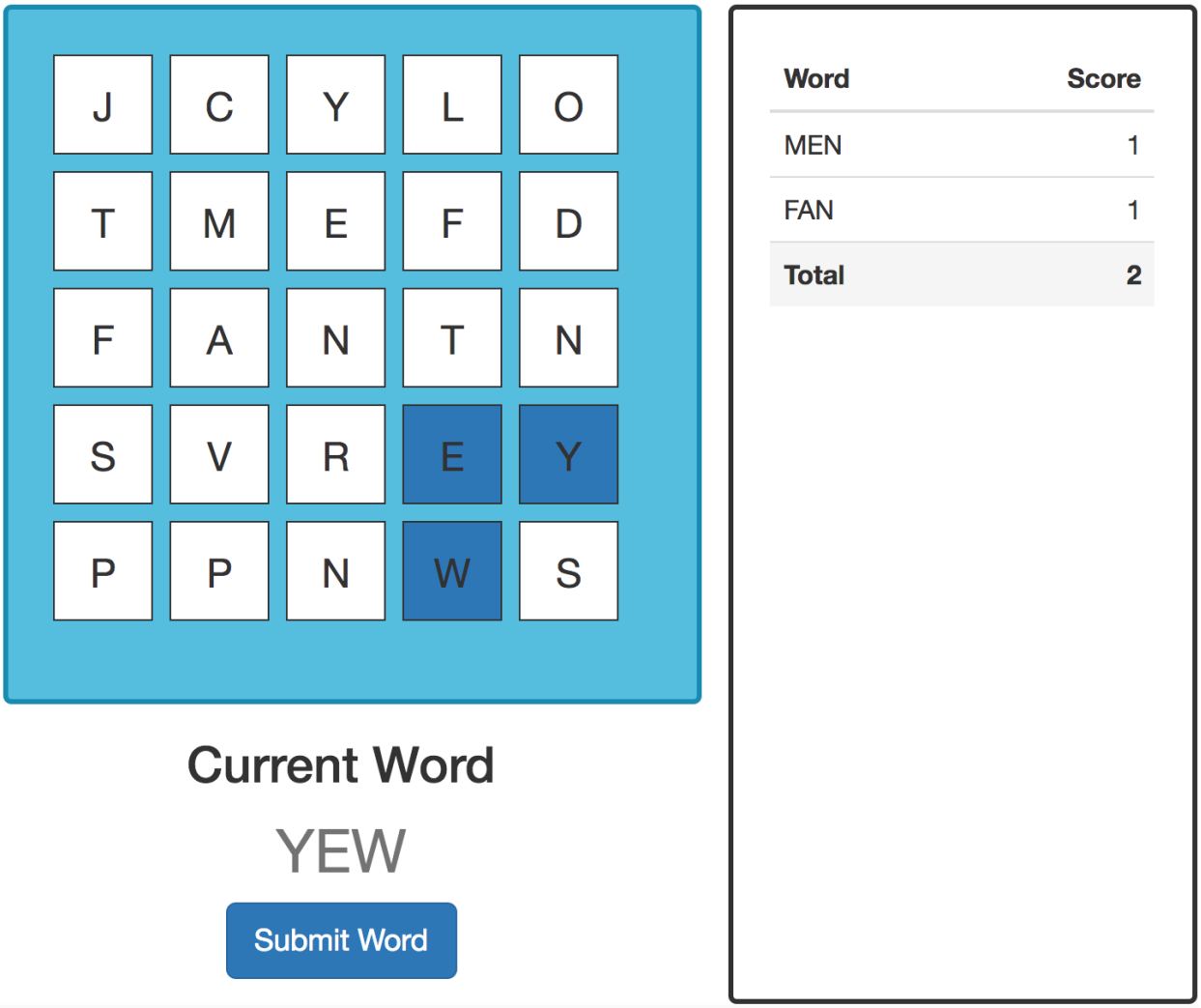
Lenda Back-end Challenge

Challenge Overview

You will be creating a server-side API meeting the requirements below. Feel free to use any Java framework and libraries that you feel are appropriate. Submit your work as a zip or tar archive, or as a shared git repository. It should be self-contained and you should include a README file with instructions on how to run the code. This exercise should not take more than xx minutes to complete, however we allocate xx hours and ask that you submit what you have completed at the xx-hour mark. This is your chance to show us at your best so we are looking for high quality code that is readable and maintainable, meets the specs, and works with the provided UI.

Requirements

This is a word-finding game suspiciously similar to https://en.wikipedia.org/wiki/Boggle.



The board is a grid of letters which are actually the faces of dice. When the game is initialized, the position and visible face of each die in the grid should be randomized. We use 25 dice with the following configurations. Each row below is a die. Each letter is a face of the die. Q will be displayed as Qu by the UI:

1. aaafrs
2. aaeeee
3. aafirs
4. adennn
5. aeeeem
6. aeegmu
7. aegmnn
8. afirsy
9. bjkqxz
10. ccenst
11. ceiilt
12. ceilpt
13. ceipst
14. ddhnot
15. dhhlor
16. dhlnor
17. dhlnor
18. eiiitt
19. emottt
20. ensssu
21. fiprsy
22. gorrvw
23. iprrry
24. nootuw
25. ooottu

Game play

- The user starts the game by clicking on a button which will invoke your back-end API that returns a new random board configuration as previously described

- The user selects squares in sequence to build up a word, which is shown in the "Current Word" section

- After the first selected square, the user can only select squares that are adjacent

horizontally, vertically, but not diagonally to the last selected square. The UI will allow the user to select any square, however, your API will need to validate that this is a valid play on the board

- The user cannot select a square that has already been selected in the current word

- The user can only unselect the last selected square in the current word

- Submit word sends the "Current Word" to your back-end API

- Duplicate words cannot be submitted

- The word must be a valid play, that is the letters must be consecutively adjacent horizontally or vertically

- The word must be a word in the dictionary file provided `dictionary.txt` (courtesy of https://github.com/jonbcard/scrabble-bot)

- Submitting a valid word clears the selected squares and "Current Word"

- The score updates when a new valid word is submitted

- Your API will return the score for a submitted valid word according to this table:

Number of Letters	3	4	5	6	7	8+
Points	1	2	3	4	5	6

API Specification

The canonical specification for your API is written in `swagger.yaml`, included in this ZIP archive, which you can load into http://editor.swagger.io/. For convenience, the documentation follows. You will create three APIs as follows:

Create a new game

`POST /game`

Description

Returns a new game with new unique ID and a board randomly scrambled according to the rules, with an empty list of words

Responses

HTTP Code	Description	Schema
200	successful operation	Game

Consumes

- application/json

Produces

- application/json

Example HTTP response

Response 200


```
{
 "id" : 5,
 "board" : [ "DOIFQ", "OGELD", "VLREN", "CRANN", "HOJPY" ],
 "score" : 4,
 "words" : [ {
  "word" : "crane",
  "score" : 3
 }, {
  "word" : "dog",
  "score" : 1
 } ]
}
```


Submit a word to the game

`POST /game/{game_id}`

Description

Accepts a word played and validates that it is a dictionary word and a valid play on the board according to the rules, and returns the score if valid, otherwise will error out

Parameters

Type	Name	Description	Schema
Path	**game_id** *required*	ID of game to return	integer (int64)

Type	Name	Description	Schema
Body	**body** *required*	Word being played (score will be ignored on input)	Word

Responses

HTTP Code	Description	Schema
200	word is a valid play and in the dictionary, response will contain score	Word
400	Word is not playable on this game board	No Content
404	Game not found	No Content
406	Word is not in the dictionary	No Content
409	Word is a duplicate	No Content

Consumes

- application/json

Produces

- application/json

Example HTTP request

Request body

```
{
  "word" : "crane",
  "score" : 3
```

}

Example HTTP response

Response 200


```
{
  "word" : "crane",
  "score" : 3
}
```


Get a game by ID

`GET /game/{game_id}`

Description

Returns a single game with that ID, including the list of words played in the game and associated scores

Parameters

Type	Name	Description	Schema
Path	**game_id** *required*	ID of game to return	integer (int64)

Responses

HTTP Code	Description	Schema
200	successful operation	Game
404	Game not found	No Content

Produces

- application/json

Example HTTP response

Response 200


```
{
  "id" : 5,
  "board" : [ "DOIFQ", "OGELD", "VLREN", "CRANN", "HOJPY" ],
  "score" : 4,
  "words" : [ {
    "word" : "crane",
```

```
   "score" : 3
  }, {
   "word" : "dog",
   "score" : 1
  } ]
}
```


Definitions

Game

Name	Description	Schema
board *optional*	array of rows on the board	< string > array
id *optional*		integer
score *optional*		integer
words *optional*	list of words already played	< Word > array

Word

Name	Description	Schema
score *optional*	ignored on input, set by the POST API	integer
word *optional*		string

UI

Open the file `challenger.html` in your web browser. The default endpoint points to a working Lenda backend so you can familiarize yourself with the game. Enter your endpoint into

the top input box, e.g. http://localhost/api/v1/game. Playing the game will now invoke your API.

CORS Considerations

If you are using an application server, it is probably best to move the files in this ZIP archive into the web root of the server so that it can serve the `challenge.html` as well as your API.

Otherwise, you may run into CORS restrictions. In that case, your APIs will need to return the appropriate headers on each endpoint, e.g.:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Content-Type
```

Archive Files

`README.pdf`	This instruction file
`axios-0.16.1.js`	JS Library for the UI
`challenge.css`	CSS for the UI
`challenge.html`	HTML for the UI, open this in your browser
`dictionary.txt`	Dictionary file for valid words, one word per line
`swagger.yaml`	API specification in Swagger format
`vue-2.3.2.js`	JS Library for the UI